executing an application component under control of an operating service, the application component having a component data state and function code for performing work on a data resource responsive to method invocations from a client, the component's work on the data resource having a work data state, the component data state initially having an initial post-creation state upon the component's creation;

providing an interface for the operating service to receive an indication from the application component that the work is complete;

maintaining the component data state in the main memory between the method invocations of the function code by the client in the absence of the indication from the application component that the work is complete; and

in response to the indication and without client action, destroying the component data state by the operating system upon a next return of the application component from a method invocation of the client, while persistently maintaining the work data state.--

REMARKS

Reconsideration of the application is respectfully requested in view of the foregoing amendments and following remarks.

Prior Art Status Of Limprecht

Claims 5-20 have been rejected under 35 U.S.C. § 103(a) as unpatentable over art described in the "Background And Summary Of The Invention" section of the specification (hereafter "Background-described art") in view of Schwartz et al., U.S. Patent 5,301,280 (hereafter "Schwartz"); Steinman, Incremental State Saving in Speedes Using C++, Proceedings of the 1993 Winter Simulation Conference, (hereafter "Steinman"); and further in view of Limprecht, Microsoft Transaction Server, Compcon '97 Proceedings, IEEE (hereafter "Limprecht"). Applicants respectfully submit that Limprecht does not qualify as prior art to the present application, obviating the rejection.

Applicants submit herewith a Declaration of Rodney Limprecht, which establishes that this reference is a publication of applicants' own invention. As such, the information contained in this reference originated from and is attributable to the inventors. This reference therefore does not constitute prior art to the present application under § 102(a). (See, MPEP § 716.10.)

Because the Limprecht reference is not prior art to the invention, the rejection based on Limprecht should be withdrawn.

Patentability Over Background Art, Schwartz, Steinman, and Bishop

Claims 1, 2 and 4 have been rejected under 35 U.S.C. § 103(a) as unpatentable over Background-described art in view of Schwartz and Steinman. Claim 3 has been rejected over these same references, and further in view of Bishop U.S. Patent Number 5,765,174 (hereafter "Bishop"). Applicants traverse the rejections.

Claim 1

Claim 1 is generally directed to enhancing the scalability of server applications by giving server application components control over the duration of their state in memory. More specifically, the claim recites,

> 1. (Amended) In a computer having a main memory, a method of enhancing scalability of server applications, comprising:
>
> executing an application component under control of an operating service, the application component having a state and function code for performing work responsive to method invocations from a client;
>
> providing an interface for the operating service to receive an indication from the application component that the work is complete;
>
> maintaining the state in the main memory between the method invocations of the function code by the client in the absence of the indication from the application component that the work is complete; and
>
> destroying the state by the operating service in response to the indication from the application component to the operating service that the work is complete and without action by the client.
>
> (Emphasis added.)

The Office asserts that the recited "destroying [application component state] in response to the indication from the application component" is taught by the Background-described art. Applicants disagree, and respectfully submit that the art (i.e., database stored procedures) described in the Background lack the recited actions of "providing an interface… to receive an indication," "maintaining the state in the main memory between method invocations… in the absence of the indication," and "destroying the state… in response to the indication." The remaining art references relied on by the Office also fail to suggest the recited actions based on this indication.

As described in the specification beginning at page 23, a server application component in the illustrated embodiment of the invention can call either "IObjectContext::SetComplete()" or

"IObjectContext::SetAbort()" before return from a client's method invocation to indicate to the run-time executive that the server application's processing work on behalf of the client within a transaction is complete. In the absence of such call, the component is maintained in memory. In response to the call, the run-time executive destroys the component's state.

By contrast, the Background-described art (i.e., stored procedures) fails to suggest that an operating service either maintains a server application component's state in memory or destroys that state based upon the absence or receipt of a "work complete" indication from the component. The Background states that user state operated upon by a stored procedure is loaded into memory during a single interaction, and stored back to secondary memory after that interaction. (See, Specification at page 2, lines 10-21.) There is no teaching or suggestion of the stored procedure indicating to an operating service whether its processing work is complete or not, and the operating service maintaining the state between method invocations or destroying the state in response to such indication.

Further, Schwartz and Steinman also fail to suggest maintaining component state in memory in the absence of the component's indication its work is complete, and destroying component state in response to the indication. Schwartz describes a communications protocol in which a connection can be disconnected by the client or broken by the server. (Schwartz, at Abstract.) However, Schwartz lacks any statement or description that a server application component make any indication to an operating service as to whether its processing work for a client is complete.

Steinman, on the other hand, describes a discrete-event simulation operating system (called "Speedes") which incrementally saves object state (i.e., saves changes in the object state) between events processed by the object. Again, Steinman fails to teach or suggest that the object indicate to the Speedes operating system whether its processing work is complete during event processing, nor the Speedes operating system maintains or destroys object state based on such indication or its absence.

Because none of the relied upon art teaches or suggests that an operating service either maintains a server application component's state in memory or destroys that state based upon the absence or receipt of a "work complete" indication from the component, claims 1-4 clearly are allowable over this art.

Claim 3

Claim 3 depends from claim 1, and further recites, “wherein the step of destroying the state comprises resetting the state of the application component to the application component’s initial post-creation state (emphasis added).”

As discussed above in connection with claim 1, the relied-upon art fails to teach or suggest that an operating service either maintains a server application component’s state in memory or destroys that state based upon the absence or receipt of a “work complete” indication from the component. Bishop also lacks any suggestion leading to this missing element.

Bishop describes a system for distributed object resource management that supports two classes of object references: strong and weak references. (Bishop, at Abstract.) When there are no strong references to an object, deletion of the object (e.g., during garbage collection) is enabled. (Bishop, at Abstract.) These references are created when a client requests cross-domain access to the object. (Bishop, at column 3, line 28 through column 4, line 13.) The system performs garbage collection by performing reference counting, and the destroying objects that lack at least one strong reference during garbage collection. (Bishop, at column 4, lines 16-67.) Bishop, however, does not teach or suggest that the object provide any indication to the system whether its processing work for the client is complete.

Moreover, the relied-upon art fails to teach or suggest the recited, “resetting the state of the application component to the application component’s initial post-creation state (emphasis added).” The Office asserts that Steinman “teaches resetting the state (restore state by calling exchange again, section 4)”. Applicants respectfully submit that Steinman’s “restore” merely reverses the “delta” or state change introduced by the object processing a single specific event, and does not teach or suggest completely resetting to an object’s “initial post-creation state” (which thus destroys all “state” formed by the object since its creation). More specifically, Steinman uses state management to correct events that may have occurred out of order due to varying processor speeds (“time accidents”). See Steinman, at 687. The Delta Exchange Method, the Rollback Queue, the Simple Assignments, and other methods discussed in Steinman, are used to return data to its intended post event state, when varying processing speeds otherwise would “produce a different result.” See Steinman, Summary at 695. The “incremental state saving” taught in Steinman is used to preserve the integrity of incremental post creation states so they may be used to repair time accidents. There is no teaching or suggestion to destroy an application component’s state by resetting to its initial post-creation state.

Also, there is no suggestion to destroy the object state via reset to initial post-creation state in the particular recited circumstance (i.e., in response to an indication that a component's processing work is complete). For Steinman to enable restoring the delta introduced by an event, Steinman teaches storing both new and old state values from before and after processing the event into a rollback queue (Steinman, at page 689). Since such rollback queue storage would increase memory usage and therefore adversely affect scalability, one would not have been led to use the event rollback of Steinman in a system that destroys component state to enhance scalability. Steinman therefore actually leads away from the claimed invention.

For these additional reasons, claim 3 should be separately allowable over this art.

CONCLUSION

The claims in their present form should now be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN CAMPBELL
LEIGH & WHINSTON, LLP

By

Stephen A. Wight
Registration No. 37,759

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
Facsimile: (503) 228-9446

(80683.1USORG)